Timer

- in C# is one of the best features available in the programming for Embedded System
- mainly used to trigger an event at a specific interval of time without any other interference
- Timer works in the background

How to use C# Timer Control?

- can control programs with Timer Control in millisecond, seconds, minutes and even in hours.
- Timer Control allows us to set Interval property in milliseconds
- By default the Enabled property of Timer Control is False

Timer Tick Event

- Timer Tick event occurs when the specified timer interval has elapsed and the timer is enabled.

Timer Elapsed Event

- Timer Elapsed event occurs when the interval elapses.
- The Elapsed event is raised if the Enabled property is true and the time interval (in milliseconds) defined by the Interval property elapses.

Timer Interval Property

- Timer Interval property gets or sets the time, in milliseconds, before the Tick event is raised relative to the last occurrence of the Tick event.

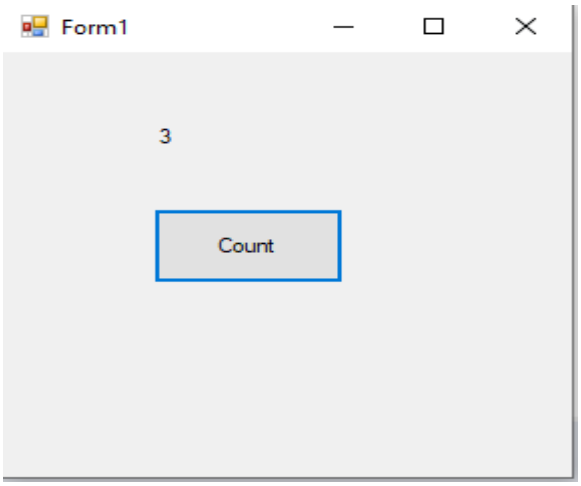
TimerCallback Delegate

- Callback represents the method that handles calls from a Timer.
- the method does not execute in the thread that created the timer
- it executes in a separate thread pool thread that is provided by the system.

Example: Count Up Timer

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ExampleTimer
{
    public partial class Form1 : Form
    {
        private int count = 0;
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            timer1 = new Timer();
            timer1.Tick += new EventHandler(timer1_Tick);
            timer1.Interval = 1000; // 1 second
            timer1.Start();
            label1.Text = count.ToString();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            count++;
            if (count == 120)
                timer1.Stop();
            label1.Text = count.ToString();
        }
    }
}
```

Output:



Serial Interfacing

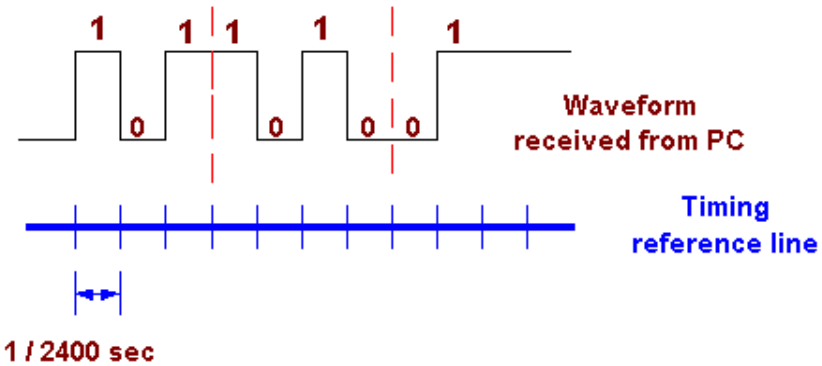
Serial communication

As the name implies, in the Serial communication you send bits of data serially i.e. one bit at a time.

Normally, the predefine rate of transfer are 2400 bits/sec. (2400bps) 56,000 bits/sec. (56 kbps), And then depending upon this rate we interpret bits boundaries.

Example of a serial waveform

You receive following waveform from serial port of your PC and it is stated that data rate is 2400bps

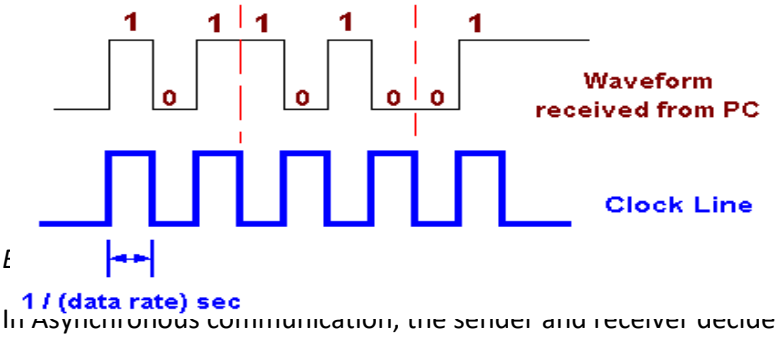


In the above waveform, the bits 10110100 represent the data, the timing reference line shows how the data is transferred serially using 2400 bps as rate.

Methods of timing used to find bit boundaries

A. Synchronous communication

In Synchronous communication, the data is transmitted by synchronizing the transmission at the receiving and sending ends using a common clock signal. Since start and stop bits are not present, this allows better use of data transmission bandwidth for more message bits and makes the whole transmission process simpler and easier.



E

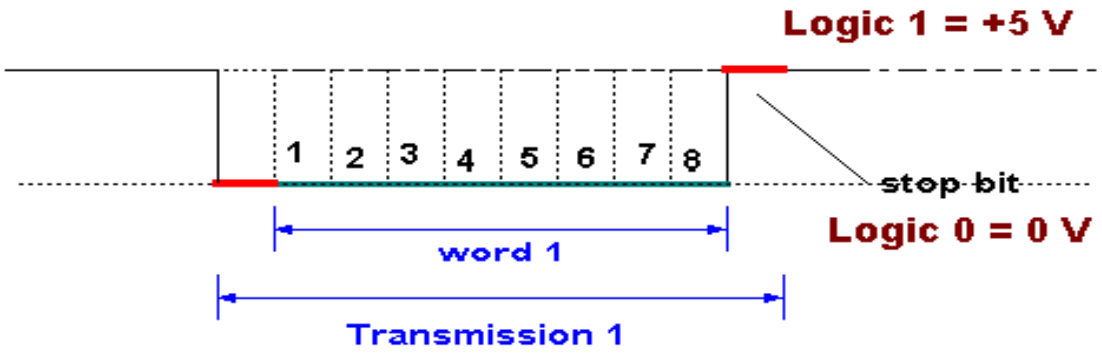
In Asynchronous communication, the sender and receiver decide a data rate before communication. They decide upon signaling used for start & stop of data transmission.

Baud Rate

Is simply "the rate of data transmission expressed in bits per second, kilo Bits per second or Megabits per second".

The sender & receiver decide upon no of bits in one data word such as 8 bits or 1byte. A first bit before transmission of data word will always be 0 (or 1) after which data bits will follow. It is called **start bit**. The last bit followed by data bits will always be 1(or 0) after which it requires start bit for transmission of next word. It is called **stop bit.**

How start bit and stop bit works?



In the above figure, ***0*** represents the start bit and ***1*** represent the stop bit. The data to transfer is ***word1***. Altogether, the bits (***transmission 1***) are transmitted serially from one device to another.

Serial Interfacing with C# and Arduino

Controlling hardware components using Microcontrollers can be done through Windows (Graphical User Interface) application developed using any programming language. Both the hardware components (IO devices) and the Windows application can send and receive data from one another. Thus, C# will be use for the windows application development and Arduino as the microcontroller.

Now, let's start with the C# program:

System.IO.Ports is a namespace contains classes for controlling serial ports.

SerialPort class is a class where methods for reading and writing date is implemented

Methods that can be used in the SerialPort class:

ReadLine(): Reads up to the NewLine value in the input buffer. If a New Line is not found before timeout, this method returns a null value.

WriteLine(string): Writes the specified string and the New Line value to the output buffer. The written output includes the New Line string.

Open(): Opens a new serial port connection.

Close(): Closes the port connection.

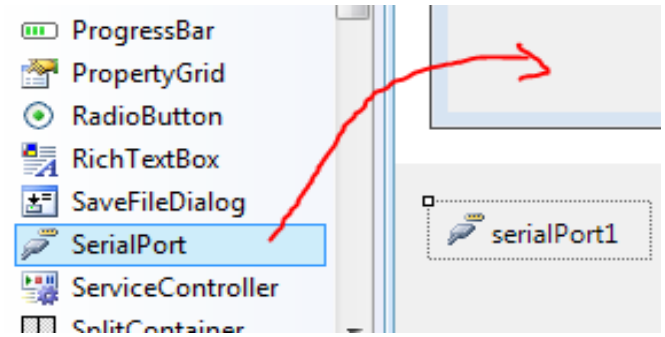
Note: There more methods you can use, please explore the SerialPort class.

Creating SerialPort Object as an Instance of the SerialPort Class:

To create a SerialPort object, all we need to do is:

```
SerialPort sp = new SerialPort ();
```

Or browse over the toolbox and drag the serial port component to the form



Properties

serialPort1 System.IO.Ports.SerialPort

(ApplicationSettings)	
(Name)	**serialPort1**
BaudRate	9600
DataBits	8
DiscardNull	False
DtrEnable	False
GenerateMember	True
Handshake	None
Modifiers	Private
Parity	None
ParityReplace	63
PortName	COM1
ReadBufferSize	4096
ReadTimeout	-1
ReceivedBytesThresh	1
RtsEnable	False
StopBits	One
WriteBufferSize	2048
WriteTimeout	-1

In all, there are seven public constructors for creating SerialPort. This constructor uses default property values.

For example, the value of DataBits defaults to 8, and StopBits defaults to 1. The default communication port will be COM1.

Properties of SerialPort class that we will use are:

BaudRate: Gets or sets the serial baud rate.

StopBits: Gets or sets the standard number of stopbits per byte.

ReadTimeout: Gets or sets the number of milliseconds before a timeout occurs when a read operation does not finish.

Note: There are many public properties, but except these three, all properties will have default values.

How to send a data?

Method to use: "Write"

with 3 overloads, writes a specified number of bytes to the serial port using data from a buffer.

Write(string text) - text: the string for output

Write(byte[] buffer,int offset,int count) - buffer: the byte array that contains the data to write to the port

Write(char[]buffer,int offset,int count) - buffer: the char array that contains the data to write to the port

Method to use: "WriteLine"

Writes the specified string and the System.IO.Ports.SerialPort.NewLine value to the output buffer

WriteLine(string text)- text: the string for output buffer

For example:

```
myport.Write("This is Great");
myport.WriteLine("This is Great");
```

How to receive a data?

Event used: DataReceived

Is an event raised each time when data is received from the SerialPort.

Method 1: using an EventHandler

EventHandler - represent the method that will handle

For example:

```
private void SerialNadawat(object sender, EventArgs e)
     {
          textBox2.Text = nadawat1.Trim();
     }

 private void serialPort1_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
     {
         //using invoke
            nadawat1 = serialPort1.ReadLine();
            this.BeginInvoke(new EventHandler(SerialNadawat));
      }
```

Method 2: Using Delegates

NET does not allow cross-threading action, to do this we used “delegates” to write to the UI thread from a non-UI thread

For example:

1. Initialize a delegate

   ```
   private delegate void SetTextDel(string text);
   ```

2. Create a method that will update a UI thread

   ```
   Private void SerialNadawat(String text)
   {
       textbox1.Text = text;
   }
   ```

3. Create the serialport DataReceived method

   ```
   private void sp_DataReceived(object sender, SerialDataReceivedEventArgs e)
   {
       String nadawat=sp.ReadLine();
       this.BeginInvoke(new SetTextDel(SerialNadawat), new object[]{nadawat});
   }
   ```

Lambda Expression

Event handlers

```
this.BeginInvoke(new EventHandler((object o, EventArgs a)
        {
//code here
        }));
```

Another example: Reading Using Events

Step 4: Reading Using Events

```
class Program
{
    static SerialPort mySerialPort;

    static void Main(string[] args)
    {
        mySerialPort = new SerialPort("COM3", 9600);

        //add our event handler
        mySerialPort.DataReceived += new SerialDataReceivedEventHandler(mySerialPort_DataReceived);

        try
        {
            mySerialPort.Open();    //open the port
        }
        catch (IOException ex)
        {
            Console.WriteLine(ex);
        }

        /*
         * any code that should run
         * when the port is not in use
         */

    }

    public static void mySerialPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
    {
        try
        {
            Console.Write(mySerialPort.ReadExisting()); //print everything in the input buffer
        }
        catch (IOException ex)
        {
            Console.WriteLine(ex);
        }
    }

}
```

The above example, creates an instance (object) of SerialPort class name *mySerialPort. mySerialPor*t uses *COM3* as the port and the rate is *9600*. Then tell the serial port what method to call when it receives data.

This is done with the following line:

mySerialPort.DataReceived += new SerialDataEventHandler(mySerialPort_DataRecieved);

mySerialPort.DataReceived - represents the method that is called to handle the event.

We specify that method by the following part

+= new SerialDataEventHandler(mySerialPort_DataReceived)

which says use the method mySerialPort_DataReceived when the event is raised.

Next we have to actually create the method:

```
public static void mySerialPort_DataRecieved(object sender, SerialDataReceivedEventArgs e)
{
  //whatever logic and read procedure we want
}
```

In the example the method works:

The mySerialPort_DataReceived method display everything what's in the buffer by the reading (use the ReadExisting()).

Sample Program:

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace Demo
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            try{
                MyPort.Items.AddRange(System.IO.Ports.SerialPort.GetPortNames());

                MyPort.SelectedIndex = 0;
                serialPort1.BaudRate = (9600);
```

```
            serialPort1.ReadTimeout = (2000);
            serialPort1.WriteTimeout = (2000);

        }
        catch(Exception ex) {
            MessageBox.Show(ex.Message);

        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        try
         {
            serialPort1.WriteLine("on");

        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);

        }
    }

    private void button2_Click(object sender, EventArgs e)
    {
        try
        {

            serialPort1.WriteLine("off");
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);

        }
    }

    private void PortBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        try
        {
            serialPort1.PortName = MyPort.Text;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);

        }
    }

    private void button3_Click(object sender, EventArgs e)
    {
        try
        {
            serialPort1.DataReceived += myport_DataReceived;

        }
```

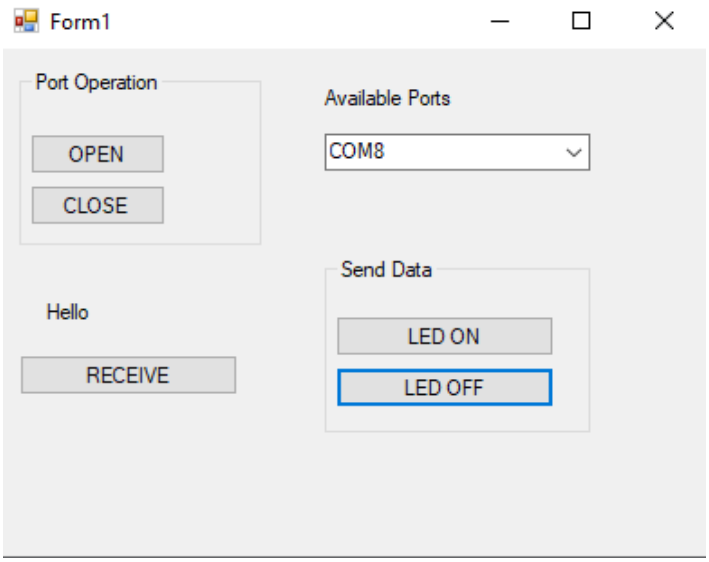
```
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);

            }
        }
        void myport_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {

            this.Invoke(new EventHandler(display_event));
        }
        //reading a data using ReadLine Method. The data is from arduino (sender) and GUI
application is the receiver

        public void display_event(Object sender, EventArgs e)
        {

            String data = serialPort1.ReadLine();
            label1.Text = data;


        }

        private void button4_Click(object sender, EventArgs e)
        {
            try
            {
                serialPort1.Open();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);

            }
        }

        private void button5_Click(object sender, EventArgs e)
        {
            try
            {
                serialPort1.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);

            }

        }

    }
}
```

Arduino:

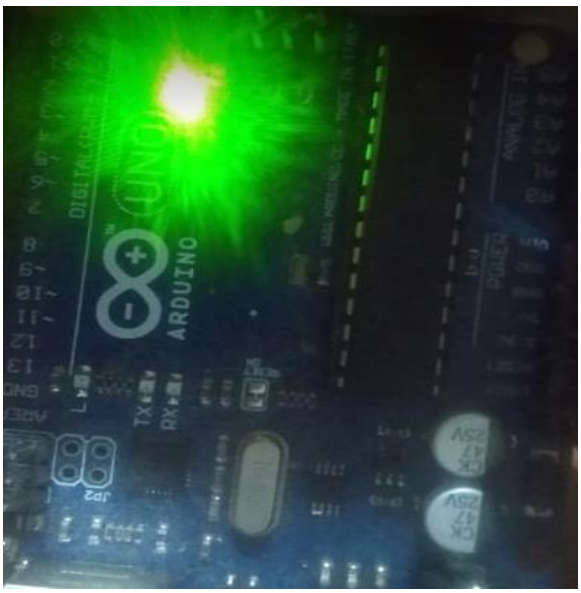
```
void setup() {
  // put your setup code here, to run once:
 Serial.begin(9600);
}
void loop() {
  // put your main code here, to run repeatedly:
  String rString;
  String s;
  while(Serial.available()){
    delay(100);
    if(Serial.available()>0){
      char x = Serial.read();
        if(isControl(x)){
     break;
      }
      rString+=x;
      }
    }
  s=rString;
  if(s=="on"){
    digitalWrite(13,HIGH);
  }
  if(s=="off"){
    digitalWrite(13,LOW);
  }
  Serial.println("Hello");
  delay(1000);
}}
```

Output:

Receive data hello



Send data Led OFF



Send data Led ON

For more example: Please click youtube link below:

https://www.youtube.com/watch?v=vHeG3Gt6STE&vl=en

https://www.youtube.com/watch?v=W4NyevEQnCI

https://www.youtube.com/?hl=en&gl=PH